

Personal Healthcare System

Submitted in partial fulfilment of the requirements for the degree of

Bachelor of Technology in Programme

by

20BCE2154

20BCE2174

20BCE2202

CSE3002 – INTERNET AND WEB PROGRAMMING

Winter Semester 2021-22: CSE3013



April, 2021

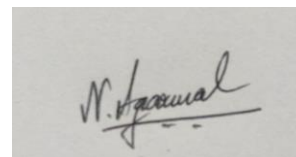
DECLARATION

I hereby declare that the thesis entitled "**Personal Health Gauging System**" submitted by me, for the award of the degree of *Bachelor of Technology in Programme* to VIT is a record of bonafde work carried out by me under the supervision of **Dr.Mythili.T**. I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

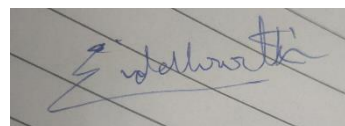
Place: Vellore

Date:29/04/2022

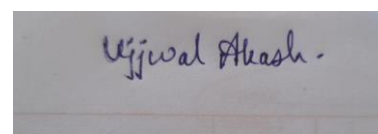
Signature of the Candidates



Nikhil Agarwal



Siddharth Gupta



Ujjwal Akash

This is to certify that the thesis entitled “**Personal Health Gauging System**” submitted by **Nikhil Agarwal(20BCE2154), Siddharth Gupta(20BCE2174), Ujjwal Akash(20BCE2202)** of SCOPE **School**, VIT, for the award of the degree of *Bachelor of Technology in Programme*, is a record of Bonafede work carried out by him / her under my supervision during the period, 01. 12. 2018 to 30.04.2019, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfils the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place: Vellore

Date: 29-04-2022

Signature of the Guide

ACKNOWLEDGEMENTS

I take immense pleasure in thanking Rd. G. Viswanathan, my beloved Chancellor, VIT University for having permitted me to carry out the project. I express gratitude to my guide, Mythili T., for guidance and suggestions that helped me to complete the project on time. Words are inadequate to express my gratitude to the faculty and staff members who encouraged and supported me during the project. Finally, I would like to thank my ever-loving parents for their blessings and my friends for their timely help and support.

Student Name

Nikhil

Siddharth Gupta

Ujjwal Akash

Executive Summary

Many people visit the doctor office on a regular basis (sometimes multiple times per week) and have no way of accurately and efficiently keeping track of their health information, such as upcoming appointments, prescriptions, symptoms they experience away from the doctor office, notes from their doctors, and height and weight measurements.

Managing all that information can be quite the challenge because it is often written on paper and scattered across multiple documents and files. Also, information often comes from multiple doctors and clinics. Plus, there is no one, central location to store the information. For example, appointments might be scheduled on a paper calendar, but the notes from that appointment are located in a separate notebook. You might experience a symptom away from the doctor office, but you don't write it down and forget to tell your doctor at your next visit. As you can see, there are many problems with the current methods for managing one's health. Keeping track of this information is a time-consuming, manual process. As a result, this information can easily be lost, misplaced, or forgotten over time.

So, it is safe to say that managing our health information is difficult. It gets even more complex and challenging when trying to manage that information for someone else (for example, a child or grandparent).

This is where BayMax comes in. BayMax is a web application that allows you to log and keep track of health information so that you can easily share and discuss that information with your doctor(s) during your next visit. It is one app that you can use to create and maintain a health journal for yourself or for another person. No more having to carry around bulky notebooks, papers, and files to each and every doctor visit. Instead, all you need to do is bring your tablet to the doctor office, open up the app, and all the relevant information is there for the doctor to see.

CONTENTS PAGE

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	DECLARATION	2
	ACKNOWLEDGEMENTS	4
	EXECUTIVE SUMMARY	5
1	INTRODUCTION	7
	1.1 Objective	
	1.2 Motivation	
	1.3 Background	
2	PROJECT DESCRIPTION AND GOALS	8
	2.1 Project Descriptions	
	2.2 Project Goals	
3	TECHNICAL SPECIFICATIONS	10
4	DESIGN APPROACH AND DETAILS	11
	4.1 Design Approach / Materials & Methods	
	4.2 Codes and Standards	
5	SCHEDULE TASKS AND MILESTONES	30
6	PROJECT DEMONSTRATION	31
7	RESULTS AND DISCUSSIONS	33
8	SUMMARY	33
9	REFERENCES	34

INTRODUCTION

1.1 OBJECTIVE

We propose to make a web based universal personal-health tracker which will allow users to log and keep track of their health so that they can easily share and discuss the information with the available doctors virtually and physically if necessary. Users will be able to create and maintain a health journal for themselves or for another like their children or parents who live in distant places. At the moment, there are many EMR systems available in the market but not many for personal medical log management. We attempt to break this flow and enable users to have control over their own medical data. The interface must be user friendly and should record all medical records, laboratory tests, prescriptions, medical contacts etc.

1.2 MOTIVATION

These years post 2020 have been intimidating to every individual on this planet. From people's jobs and the education of their children, everything has been affected by Covid-19 pandemic. Parents have struggled not only to run the household but also to keep themselves and their families healthy and away from any illness. Many people visit the doctor's office on a regular basis and have no way of precisely and competently keeping track of their health information, such as appointments, prescriptions, symptoms, and notes from their doctors which is highly important at this point of time. Managing this information can be quite the challenge because it's often written on paper, scattered across multiple documents and files. We propose to make a web based universal personal-health gauging system which will allow users to log and keep track of their health so that they can easily share and discuss the information with the available doctors virtually and physically if necessary. Users will be able to create and maintain a health journal for themselves or for another like their children or parents who live in distant places.

1.3 BACKGROUND

Due to outbreak of covid people are facing lot of issues whether it is running a household or maintaining health status. Also, COVID-19 has made people more concerned about their health especially older ones. In order to keep track of their health a website is needed which is competently keeping track of their health information, such as appointments, prescriptions, symptoms, and notes from their doctors which is highly important at this point of time. The main concern is that the people who are living far by can keep track of the health of their loved ones. This will also show the doctors available virtually present and offline in the given slots. After observing the situation, we have decided to make a website which will competently keeping track of their health information, such as appointments, prescriptions, symptoms, and notes from their doctors which is highly important at this point of time.

PROJECT DESCRIPTION AND GOALS

2.1 PROJECT DESCRIPTION

How the app is built

On the front-end side, this project is built using React, which is an open-source Javascript library developed at Facebook specifically for the task of developing user interfaces. React relies on a component-based architecture where elements of the user interface are broken into small chunks of code called components. Material UI is a css framework that helps with building these components. Material UI is a React component library that implements Google's material design.

On the back end side, this project uses MongoDB, Node, Express, Mongoose ORM (Object Relational Mapper), Passport, and various third party packages. Node, MongoDB, and Mongoose are used to query and route data in the app. Express is the backend web framework used for this app. Passport is authentication middleware for Node.js (that is, the technology used to log users into the app).

App workflow Authentication

When you first visit the app, you will be prompted to log in (if you have already created an account) or sign up (if you are a new user). To sign up, you will need to provide an email address as well as create a username and password. Authentication is required because it ensures that a user's health information is protected and blocked from other users.

Home

After you authenticate, you are taken to the Home/Landing page. From this page, you can navigate to all the different pages with the app to keep track of your health information.

My health logs

The My health log page is where you can manually record doctor appointment information, including doctor being seen, date of visit, reason for visit, weight, height, and visit notes. You can track your height and weight measurements over time from the Charts page.

My symptom journals

The My symptom journal page is a log of symptoms that you can share with your doctor at your next doctor visit. You can record symptoms you experience away from the doctor office, the time when the symptom occurred, and add comments to each entry. You have a list of symptoms to choose from (for example, Dizzy, Shortness of Breath, Fainted, Swelling, Heart Fluttering, Fatigue, Other). You have a text field where you can enter more information about a symptom and list any useful information that will help you remember details of the symptom when you see your doctor at your next visit.

Appointments

The Appointments page lists your upcoming appointments and related information in table format. You can add and remove appointments from this page.

Prescriptions

The Prescriptions page is where you can enter information about medications prescribed by your doctor, including prescription name, name of doctor who prescribed the medication, date prescribed, number of tablets, and directions for use.

Doctors and clinics

The Doctors and clinics page has contact information for each of your doctors, such as doctor name, email address, and office location. You can also add contact information for each clinic. After you add your doctors and clinics, you can associate these doctors and clinics with other information, such as appointments, in the app.

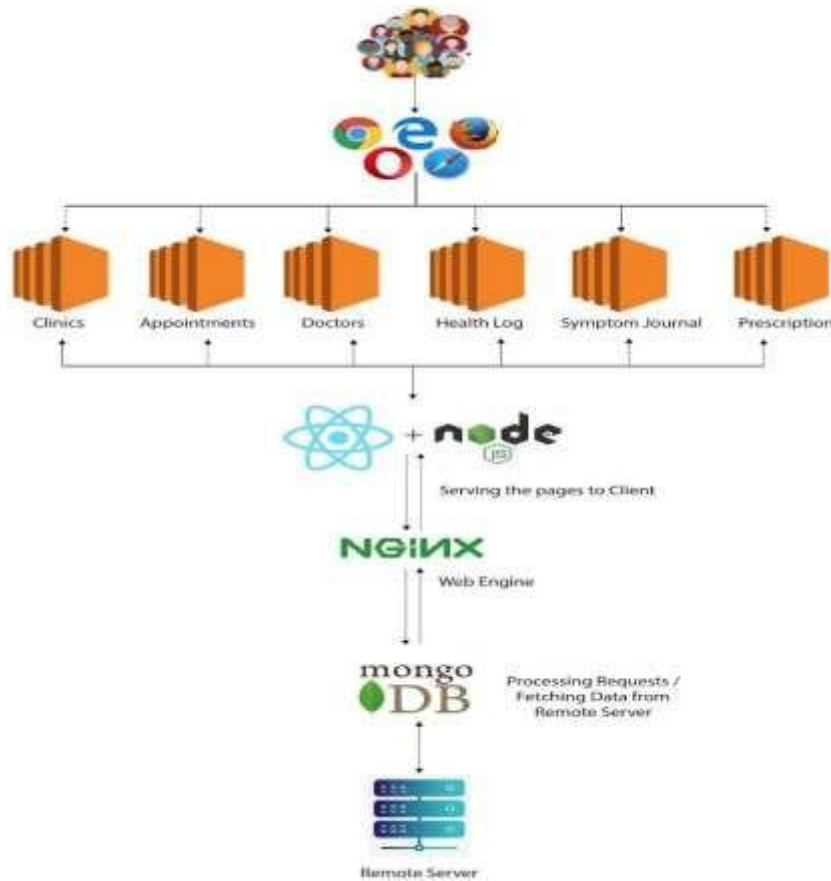
Charts

The Charts page includes two-line graphs. The top graph displays your weight data over time. This data is pulled from the weight measurements you entered on the My health log page. The second graph is similar. It displays your height data over time.

2.2 GOALS

Our main aim for making this website is that it helps the people especially who are having chronic conditions during COVID-19 because those who are suffering have more chances of getting affected to it easily. It will help elderly people to maintain their records and also their families who are living distant places can access their records and have notes of doctor as well as their medicines on the regular basis. During the COVID time people have started prioritizing their health more. They could check every day but there is no place where they can track and store their daily inputs. By the help of this website, they can track and store their daily inputs.

3. TECHNICAL SPECIFICATIONS



Back-end technologies

- Node.js (<https://nodejs.org/en/>)
- MongoDB (<https://www.mongodb.com/>)
- Express (<http://expressjs.com/>)
- Mongoose ORM (<http://mongoosejs.com/>)
- JavaScript
- AWS S3 Buckets (<https://aws.amazon.com/s3/>)

Front end technologies

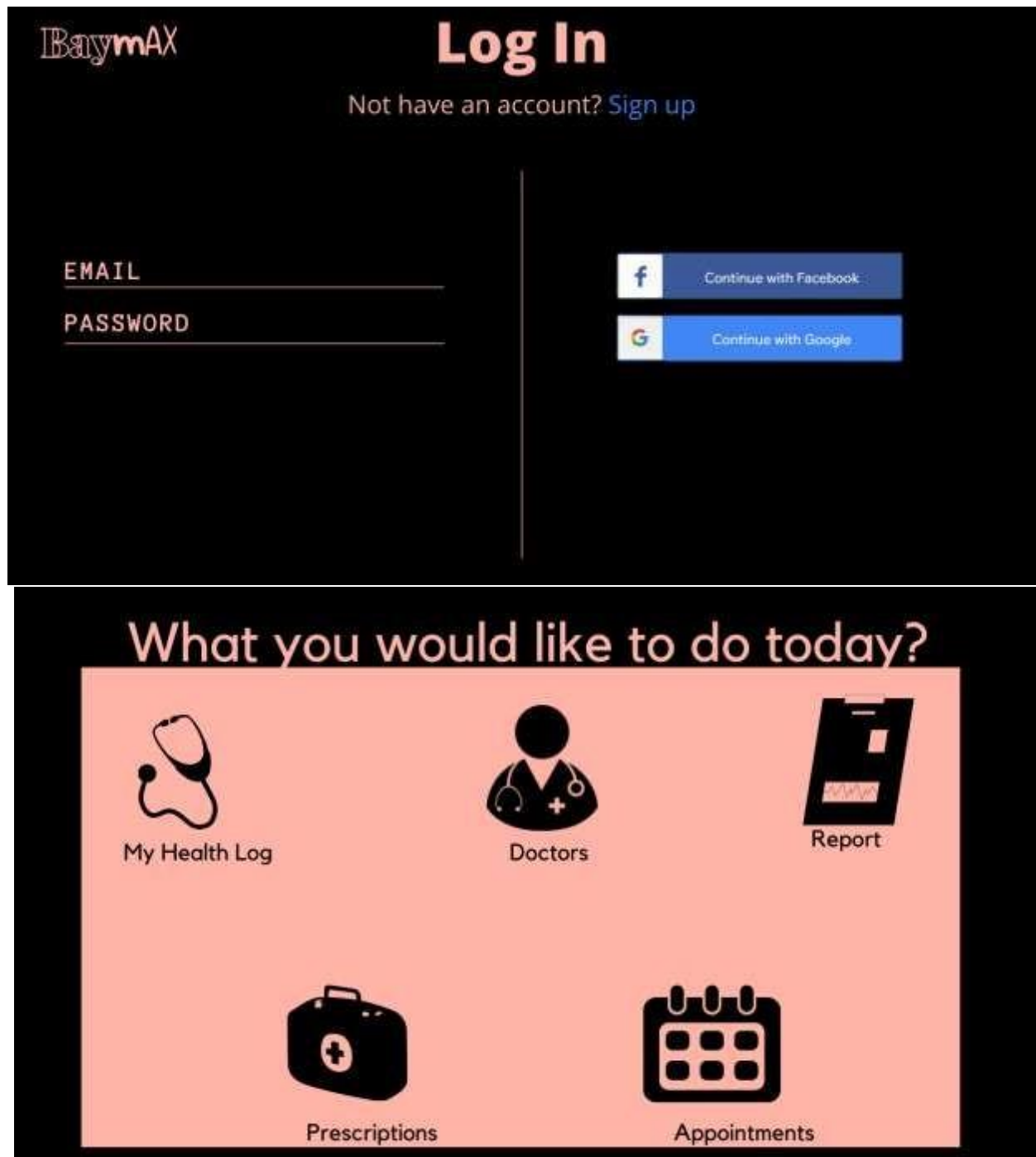
- HTML
- CSS
- JavaScript
- React (<https://reactjs.org/>)
- Material UI Next (<https://material-ui-next.com/>)
- Fusion Charts (<https://www.fusioncharts.com/>)

4. DESIGN APPROACH AND DETAILS (as applicable)

4.1 Design Approach / Materials & Methods

Firstly, we made 2 prototypes on Canva in order to get better idea what the client needs

Prototype 1



Prototype2

Welcome to Baymax

LOGIN

Enter your HealthTracker credentials to LOG IN or click SIGN UP to create an account.



[SHOW PASSWORD](#)

[LOG IN](#) [SIGN UP](#)

What would you like to do today?

My health log



My symptom journal



Appointments



My prescriptions



Doctors and clinics



Charts



After discussion we decided to move ahead with the second prototype.

4.2 Codes and Standards

CODES:

Sample Source Code (Only for Appointments module)–

a. Frontend of Appointments Module with React.js

```
// Importing React since we are using React.
import React, { Component } from "react"; //
Importing the AppointmentsForm component.
import AppointmentsForm from './AppointmentsForm'; //
Importing the AppointmentsList component.
import AppointmentsList from './AppointmentsList';
// Import API
import AppointmentAPI from '../utils/AppointmentAPI';
import DoctorsAPI from '../utils/DoctorsAPI'; import
ClinicsAPI from '../utils/ClinicsAPI'; // Import UI
components from material-ui-next. import { withStyles }
from 'material-ui/styles';
import Table, { TableBody, TableCell, TableHead, TableRow } from 'material-ui/Table'
import Paper from 'material-ui/Paper';
import Typography from 'material-ui/Typography';
import Grid from 'material-ui/Grid'; //
Import Sidebar component.
import Sidebar from '../Components/Sidebar';
// Importing Navbar component. import
NavBar from '../Components/AppBar';
```

```
//Style
```

```
const styles = theme => ({
  root: theme.mixins.gutters({
    marginTop: theme.spacing.unit * 3,
    borderStyle: 'solid',
    borderWidth: 4,
    borderColor: '#33658A',
    display: 'block',
```

```

        overflowX: 'auto',
        maxWidth: '80%',
      }),
      table: {
        minWidth: 700,
        tableLayout: 'auto',
        display: 'block',
        width: '100%',
        overflowX: 'auto',
      },
      tableWrapper: {
        overflowX: 'auto',
      },
      row: {
        '&:nth-of-type(odd)': { backgroundColor:
          'theme.palette.background.default',
        },
      },
      appFrame: {
        zIndex: 1,
        overflow: 'hidden',
        position: 'relative',
        display: 'flex',
        width: '100%',
      },
      content: {
        flexGrow: 1, backgroundColor:
          '#86BBD8',
        padding: theme.spacing.unit * 3,
      },
      heading: {
        marginTop: 40,
      },
    });

```

```

class Appointments extends Component { state
= {
  appointmentName: '',
  appointmentDoctor: '',
  appointmentDate: '',
  appointmentTime: '',
  appointments: [],
  doctors: [],
  clinics: [],
  appointmentNameError: '',
  appointmentDoctorError: '',
  appointmentDateError: '',
  appointmentTimeError: '',

```

```

};
// When the component mounts, load all appointments and save them to
this.state.appointments.
componentDidMount() {
  this.loadAppointments();
  this.loadDoctors();
  this.loadClinics(); }

// Loads all appointments and saves them to this.state.appointments. loadAppointments
= () => {
  AppointmentAPI.getAppointments()
    .then(res => this.setState({
      appointments: res.data}))
    )
    .catch(err => console.log(err));
};

// Deletes an appointment from the database with a given id, then reloads appointments from
the db
deleteAppointment = id => {
  AppointmentAPI.deleteAppointment(id)
    .then(res => this.loadAppointments())
    .catch(err => console.log(err));
};

//Loads all doctors and saves them to this.state.doctors.
loadDoctors = () => {
  DoctorsAPI.getDoctors()
    .then(res => this.setState({
      doctors: res.data }))
    )
    .catch(err => console.log('getting doctors did not work: ', err));
};

//Loads all clinics and saves them to this.state.clinics.
loadClinics = () => {
  ClinicsAPI.getClinics()
    .then(res => this.setState({
      clinics: res.data }))
    )
    .catch(err => console.log('getting clinics did not work: ', err));
};

// Keep track of what user enters for appointment name so that input can be grabbed later.
// If form validation error is showing, remove error from page when user starts typing.

```

```

this.setState({
  appointmentName: event.target.value,
  appointmentNameError: "", formSuccessMessage:
  "",
}); }

// Keep track of what user selects for doctor so that input can be grabbed later.
// If form validation error is showing, remove error from page when user starts typing.
handleAppointmentDoctorChange = (event) => {
  this.setState({
    appointmentDoctor: event.target.value,
    appointmentDoctorError: "",
    formSuccessMessage: "",
  }); }

// Keep track of what user types into appointment date input field so that input can be
grabbed later.
// If form validation error is showing, remove error from page when user starts typing.
handleAppointmentDateChange = (event) => { this.setState({
  appointmentDate: event.target.value,
  appointmentDateError: "",
  formSuccessMessage: "",
}); }

// Keep track of what user types into appointment time input field so that input can be
grabbed later.
// If form validation error is showing, remove error from page when user starts typing.
handleAppointmentTimeChange = (event) => { this.setState({
  appointmentTime: event.target.value,
  appointmentTimeError: "",
  formSuccessMessage: "",
}); }

handleFormSubmit = event => {
  event.preventDefault();

  // If appointment name field is empty when user submits form, show error.
  if (this.state.appointmentName === "") {
    this.setState({
      appointmentNameError: "Enter a name for the appointment."
    })
  }
}

```



```

// If the appointment doctor field is empty when user submits form, show error.
if (this.state.appointmentDoctor === "") {
  this.setState({ appointmentDoctorError: "Select the doctor associated with the
  appointment from the
drop-down list."
  })
}

// if the appointment date field is empty when user submits form, show error. if
(this.state.appointmentDate === "" || this.state.appointmentDate === "mm/dd/yyyy") {
  this.setState({
    appointmentDateError: "Use the date picker to select the date of the
    appointment." }) }

// if the appointment time field is empty when user submits form, show error. if
(this.state.appointmentTime === "") {
  this.setState({
    appointmentTimeError: "Use the time picker to select the time of the appointment in
    HH:MM AM/PM format."
  })
}

else {
  //Save appointment to database if all fields are filled out.
  // Show form success message to user.
  AppointmentAPI.saveAppointment({ appointmentName:
  this.state.appointmentName,
  doctor: this.state.appointmentDoctor,
  date: this.state.appointmentDate,
  time: this.state.appointmentTime,
  })
  .then(res => this.loadAppointments()) .catch(err
  => console.log(err));

  this.setState({ formSuccessMessage:
    `${this.state.appointmentName} with Dr.
    ${this.state.appointmentDoctor} on ${this.state.appointmentDate} added successfully!`, });

  // Clear form document.getElementById('appointment-
  form').reset(); }
};

```



```

    </div>
  </Paper>
</Grid>
</Grid>

<Grid container spacing={24} className={classes.heading}>
  <Grid item xs={12}>
    <AppointmentsForm
      doctors={this.state.doctors} clinics={this.state.clinics}
      handleFormSubmit={this.handleFormSubmit}
      handleAppointmentNameChange={this.handleAppointmentNameChange}
      handleAppointmentDoctorChange={this.handleAppointmentDoctorChange}
      handleAppointmentDateChange={this.handleAppointmentDateChange}
      handleAppointmentTimeChange={this.handleAppointmentTimeChange}
      handleAppointmentClinicChange={this.handleAppointmentClinicChange}
      appointmentNameError = {this.state.appointmentNameError}
      appointmentDoctorError = {this.state.appointmentDoctorError}
      appointmentDateError = {this.state.appointmentDateError}
      appointmentTimeError = {this.state.appointmentTimeError}
      formSuccessMessage = {this.state.formSuccessMessage} />
    </Grid>
  </Grid>
</div>
</div>
</main>
</div>
];
}
}

// Exporting the Appointments component
// so that the App.js file can use/render the Appointments page.
export default withStyles(styles)(Appointments);

```

b. Schema of Appointments Module

```

const mongoose = require('mongoose');

// Save a reference to the Schema constructor const Schema = mongoose.Schema;

// more about RegEx Patterns here https://www.regexbuddy.com/regex.html

// To create an appointment, we will require the following: date, time // doctor, clinic, and name
// of appointment.

```

```

// new AppointmentSchema object for login purposes
const AppointmentSchema = new Schema({
  date: {
    type: Date,
    trim: true,
    required: 'Date name is required',
  }, time:
  {
    type: String, trim:
    true,
    required: 'Address is required',
  },
  // this is populated by dropdown
  doctor: {
    type: String, trim:
    true,
    required: 'Address is required',
  },
  //this is reason for visit
  appointmentName: {
    type: String,
    trim: true, },
  // `date` must be of type Date. The default value is the current date
  userCreated: {
    type: Date, default:
    Date.now,
  },
});

// This creates our model from the above schema, using mongoose's model method
const Appointment = mongoose.model('Appointment', AppointmentSchema);

// Export the Appointment model module.exports
= Appointment;

```

c. Database Controller of Appointments Module

```

const db = require('./models');

// Defining methods for the
appointmentsController module.exports = {
  findAll: function(req, res) { db.Appointment
    .find(req.query)
    .then(dbModel => res.json(dbModel))
    .catch(err => res.status(422).json(err));
  }
};

```

```

    },
    create: function(req, res) { db.Appointment
      .create(req.body)
      .then(dbModel => res.json(dbModel))
      .catch(err => res.status(422).json(err));
    },
    findById: function(req, res) {
      db.Appointment
        .findById(req.params.id)
        .then(dbModel => res.json(dbModel))
        .catch(err => console.log('the findById appointment is not working in
appointmentscontroller.js error: ' + err));
      //res.status(422).json(err);
    },
    update: function(req, res) {
      db.Appointment
        .findOneAndUpdate({ _id: req.params.id }, req.body)
        .then(dbModel => res.json(dbModel))
        .catch(err => console.log('the update appointment is not working in
appointmentscontroller.js error: ' + err));
      //res.status(422).json(err);
    },
    remove: function (req, res) {
      db.Appointment
        .findById({ _id: req.params.id })
        .then(dbModel => dbModel.remove())
        .then(dbModel => res.json(dbModel))
        .catch(err => console.log('the remove appointment is not working in
appointmentscontroller.js error: ' + err));
      //res.status(422).json(err); },
  };
};

```

d. Router of Appointments Module with express.js

```

//these are required
const router = require("express").Router();
const appointmentsController = require("../controllers/appointmentsController");
const isAuthenticated = require('../isAuthenticated')

module.exports = function(passport){ //
  Matches with "/api/appointments"
  router.route("/")
    .get(isAuthenticated, appointmentsController.findAll)
    .post(appointmentsController.create);

  // Matches with "/api/appointments/:id"
  router.route("/:id")
    .get(isAuthenticated, appointmentsController.findById)
    .put(appointmentsController.update)
    .delete(appointmentsController.remove);
}

```

```
return router; }
```

e. Website's App.js with all components

```
// Importing React since we are using React. import React, { Component }
from 'react'; // Importing React Router to add page routes. import {
BrowserRouter as Router, Route, Switch } from 'react-router-dom'; //
Importing material-ui theme.
import { MuiThemeProvider, createMuiTheme } from 'material-ui/styles';
// Importing css import
'./App.css';
// Importing Footer component. import
Footer from './Components/Footer'; //
Importing Home page component. import
Home from './containers/Home';
// Importing the My symptom journal page component. import
SymptomJournal from './containers/SymptomJournal'; //
Importing the DoctorList page component.
import DoctorList from './containers/DoctorsList'; //
Importing the Appointments page component.
import Appointments from './containers/Appointments';
// Importing the My health log page component.
import MedLog from './containers/MedLog';
// Importing the Prescriptions page
import Prescriptions from './containers/Prescriptions';
// Importing the Charts page
import Charts from './containers/Charts';
// Importing the Login page
import Login from './containers/Login';
// Importing the Signup page
import Signup from './containers/Signup';
// Importing the 404 page import NoMatch
from './containers/NoMatch';

// App theme customization.
const theme = createMuiTheme({ palette: { type: 'light', // Switching the dark
mode on is a single property value change.
},
typography: {
```

```

    // In Japanese the characters are usually
    larger. fontSize: 18,
  },
});

class App extends Component {

  state = {
    currentlyLoggedInUser:
    null }
  setUser = userId => {
    this.setState ({ currentlyLoggedInUser: userId})
  }

  render
  () {
    retur
    n [
      <MuiThemeProvider theme={theme}>
        <Router>
          <div>
            <Switch>
              <Route exact path="/" render={props => <Login
{...props} setUser={this.setUser}></Login>}/>
              <Route exact path="/signup" component={Signup} loggedInUser
              =
              {this.state.currentlyLoggedInUser}/>
              <Route exact path="/home" component={Home} />
              <Route exact path="/symptoms" render={props => <SymptomJournal
{...props}></SymptomJournal>
              loggedInUser={this.state.currentlyLoggedInUser}
            />
              <Route exact path="/doctors" component={DoctorList} />
              <Route exact path="/appointments" component={Appointments} />
              <Route exact path="/log" component={MedLog} />
              <Route exact path="/prescriptions" component={Prescriptions} />
              <Route exact path="/charts" component={Charts} />
              <Route component={NoMatch} />
            </Switch>
          </div>
        </Router>
        <Footer />
      </MuiThemeProvider>,
    ];
  }
}

```

```
// Exporting App component so that index.js can access it and render the components to
```

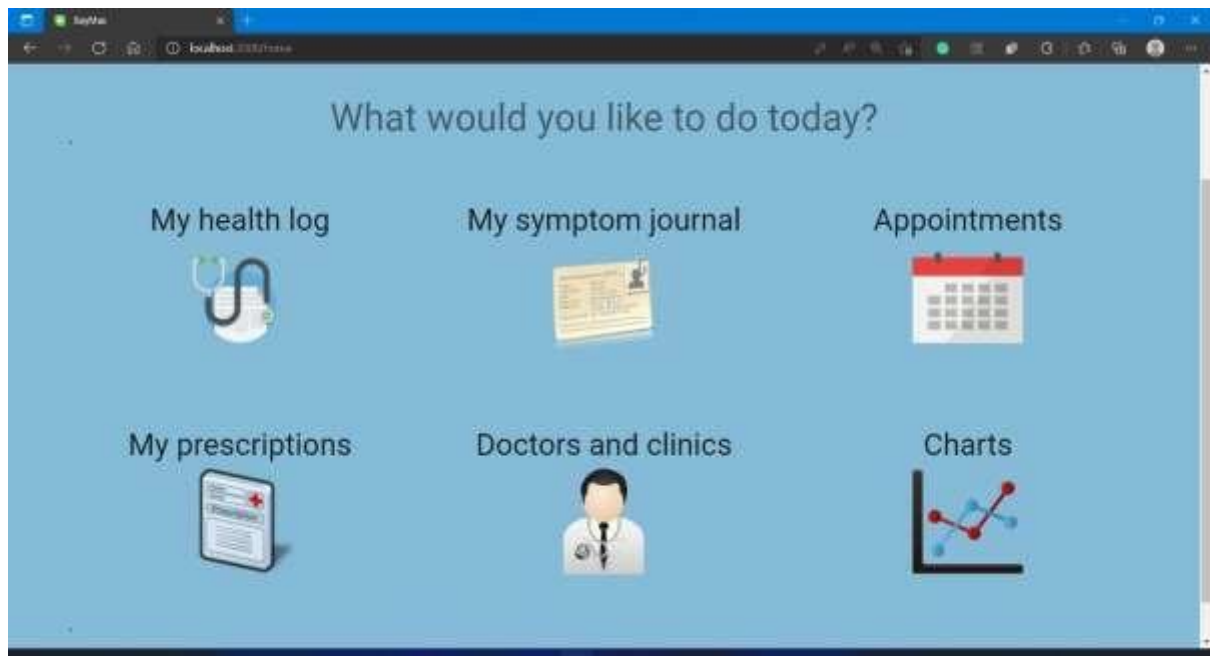
```
the  
page.  
export default App;
```

SAMPLES:

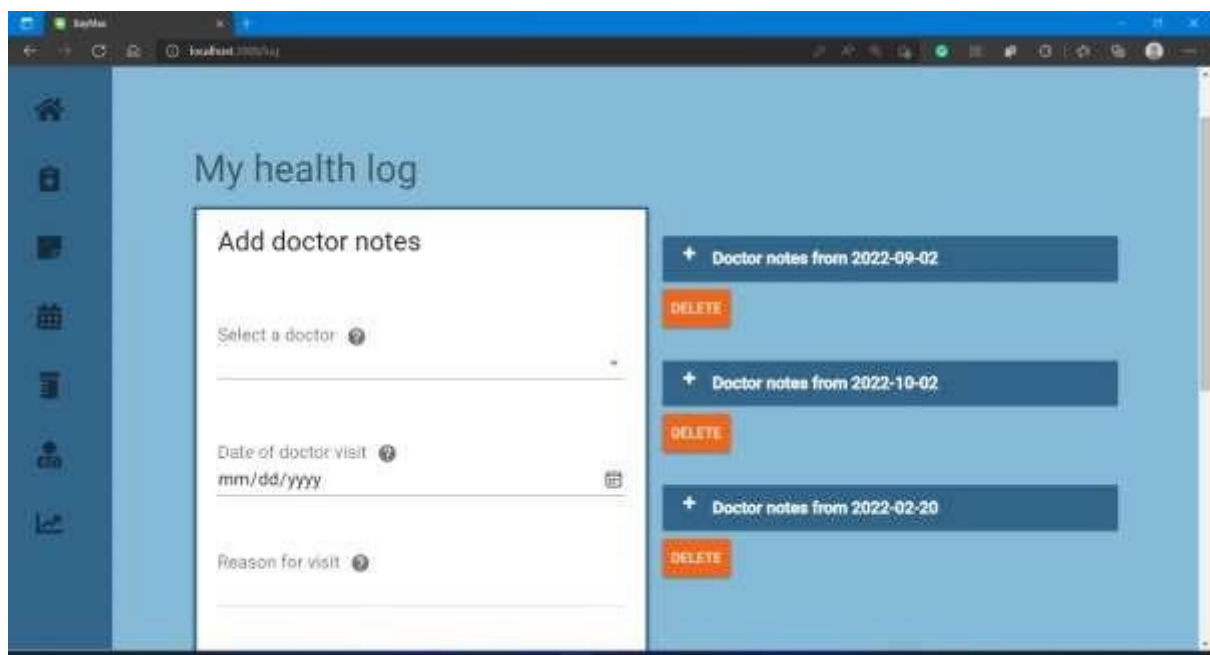
LOGIN:



MENU:



MY HEALTH LOG PORTAL:



Visit notes: 1/5

DELETE

+ Doctor notes from 2022-10-02

DELETE

+ Doctor notes from 2022-02-20

Doctor: lakasfi

Reason for visit: headache

Height (inches): 80

Weight (pounds): 120

Visit notes: 1/1

Reason for visit

Height (inches)

Weight (pounds)

Visit notes

ADD NOTES

MY SYMPTOM JOURNAL PORTAL:

My symptom journal

Add a symptom

Select symptom type

Day symptom occurred

Time symptom occurred (HH:MM AM/PM)

Additional information you want to share with your doctor

Symptom: Dizzy

Date: 2022-03-14

Time: 23:31

More info: head ache and dizziness

REMOVE

Symptom: Shortness of breath

Date: 2022-03-17

Time: 18:00

More info: xyz

REMOVE

Symptom: Fatigue

Date: 2022-04-16

Time: 19:56

More info: pqr

PRESCRIPTIONS PORTAL:

The image displays two screenshots of the BayMax Prescriptions Portal. The top screenshot shows the 'Add a prescription' form and a list of existing prescriptions. The bottom screenshot shows the full form with fields for Name, Doctor, Date, Amount, and Directions, along with an 'ADD PRESCRIPTION' button.

BayMax LOGOUT

Prescriptions

Add a prescription

Name

Doctor who prescribed

Existing Prescriptions:

- Name:** lansiprosole
Prescribing doctor:
Date prescribed: 10/16/16
Amount: 10mLs
Directions for use: Take half hour before breakfast and dinner.
REMOVE
- Name:** singular
Prescribing doctor:
Date prescribed: 10/16/17
Amount: 5mL, disolvable tablet
REMOVE

Form Fields:

Doctor who prescribed

Date prescribed mm/dd/yyyy

Dose amount

Directions for use

ADD PRESCRIPTION

APPOINTMENTS PORTAL

The screenshot shows a web application titled "Appointments" with a sidebar containing icons for home, appointments, doctors, clinics, and a profile. The main content area displays a table of appointments and a form to add a new one.

Name	Doctor	Date	Time (HH:MM)	
knee injury for nikhil agarwal	gupta	18 Apr 2022	00:02	

Add an appointment

Appointment name

Select a doctor

The screenshot shows a web application titled "Appointments" with a sidebar containing icons for home, appointments, doctors, clinics, and a profile. The main content area displays a form to add a new doctor and a list of clinics.

Select a clinic

Phone number

ADD DOCTOR

Doctor: Siddharth gupta

Clinic: Apollo

Phone: 7893425679

DELETE

Add a clinic

Name

Address

Clinics

Clinic: Apollo

Address: bilaspur

DOCTORS AND CLINIC PORTAL

The image displays two screenshots of a web application titled "BayMax" with a "LOGOUT" link in the top right corner. The application is running on a browser at the URL "localhost:3000/doctors".

The first screenshot shows the "Doctors and clinics" section. It features a sidebar with navigation icons and a main content area with two panels:

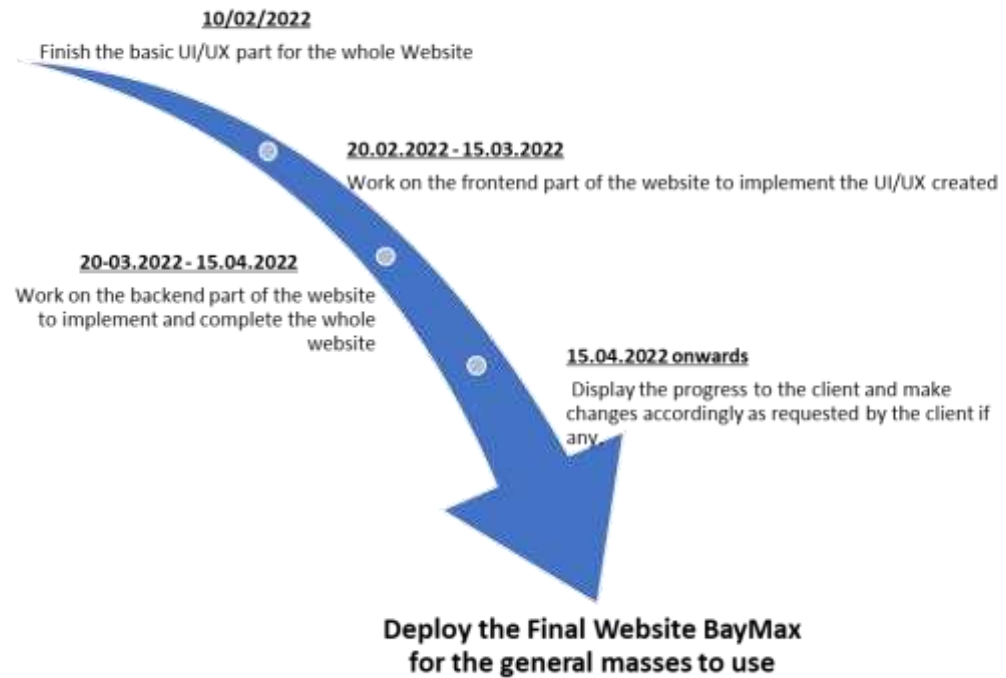
- Add a doctor:** A form with input fields for "First name", "Last name", and a "Select a clinic" dropdown.
- Doctors list:** A list of doctors with details for "ujjwal aakash" (Clinic: Apollo, Phone: 9823477680) and "Siddharth gupta". Each entry has a "DELETE" button.

The second screenshot shows the "Clinics" section. It also features a sidebar and a main content area with two panels:

- Add a clinic:** A form with input fields for "Name", "Address", "City", "State", and "Zip code".
- Clinics:** A list of clinics with details for "Apollo" (Address: bilaspur, bilaspur, AZ 495555, Phone: 9923552344). It includes a "DELETE" button.

SCHEDULE, TASKS AND MILESTONES

Schedule



PROJECT DEMONSTRATION

client

public: The public folder contains the index.html file. This HTML file is a template. The file is empty. So, if you open it directly in a browser, you will get an empty page. Rather than placing the HTML code directly in index.html, this app uses a React component-based architecture to create, build, and render UI components to the page.

src: The src folder is where the React app components reside.

assets: Contains the images/icons used in the app.

index.js: The index.js file is the top-level file of the React app. In index.js, the App.js file is imported, and the ReactDOM.render method is used to render App.js to the page.

App.js: The App.js file is where the app components are imported and rendered, such as the navigation bar, footer, and various pages.

Components: The Components folder is where the app components that are reused across the app are located. Each file represents a separate component. For example, AppBar.js is the top navigation bar component.

containers: Holds all the pages of the app and the child components within those pages. For example, inside of the containers folder, there is an Appointments folder. The Appointments folder contains a top-level parent container/page called Appointments.js that has two child containers (AppointmentsForm.js and AppointmentsList.js).

utils: Contains all the axios requests used to get health information from the database.

App.css and index.css: The external css stylesheets for the app.

package.json: Lists the project dependencies and their version numbers.

yarn.lock: Dependency tree for the project. Lists all the client dependencies and their versions.

Controllers: The controllers are the routes that are used to pass information to and from the view and model objects. models: The models define the database schema or structure of the database.

routes: These files are the key to how the back end and front end can communicate with each other. They give the server a map of how to respond when users visit or request data from various URLs. scripts

build.js: Run yarn build in the project root directory to create a production build of the app, which you can use to deploy the app to Heroku. seedDB.js: Run yarn seed to populate your development database with information. start-client: Script used to start the React development server.

.eslintrc.js: List of rules and their definitions for ESLint.

.gitignore: Anything listed inside this file (for example, node_modules) will not be tracked by GitHub or Heroku when code is committed.

package.json: Lists the project dependencies and their version numbers. It also contains various scripts to start the server, create a production build, seed the database, etc.

Procfile: This file tells Heroku to run the server file with node once it's built. server.js:

This file does the following:

Defines and requires the dependencies, including express, body-parser, and mongoose.

Sets up the Express server to handle data parsing using body-parser.

Points the server to the API routes, which gives the server a map of how to respond when users visit or request data from various URLs.

Defines the port the server is listening on.

Starts the server and React server.

Allows the app to serve static content.

Sets up Passport, which allows the user to authenticate/log in to the app.

Uses Mongoose (orm) to connect to MongoDB, which allows us to have access to the MongoDB commands to perform various operations on the database.

yarn.lock: Dependency tree for the project. Lists the project dependencies and their versions.

RESULT & DISCUSSION

This website is already promoting a lot of interaction between the users. The community page can be expanded further by creating a space for the users to track their health experiences as well. This can include blood sugars or fevers etc. Apart from these ideas, a few more features can be incorporated to refine the website, like checking blood pressure or including a few more options while chatting, providing recommended medicines to a particular user based on their browser history, etc. This website is designed to build a community by gathering people from all parts of the world to keep themselves healthy and lead a good life.

SUMMARY

We propose to make a web based universal personal-health tracker which will allow users to log and keep track of their health so that they can easily share and discuss the information with the available doctors virtually and physically if necessary. Users will be able to create and maintain a health journal for themselves or for another like their children or parents who live in distant places. At the moment, there are many EMR systems available in the market but not many for personal medical log management. We attempt to break this flow and enable users to have control over their own medical data. The interface must be user friendly and should record all medical records, laboratory tests, prescriptions, medical contacts etc.

References

Journals –

1. http://www.ijircce.com/upload/2018/april/67_A%20Review.pdf
2. https://www.theseus.fi/bitstream/handle/10024/153461/Markus_Keinanen.pdf?sequence=4
3. http://www.ijircce.com/upload/2018/april/66_Analysis.pdf

pdf Websites -

4. <https://www.geeksforgeeks.org/mern-stack/>
5. <https://medium.com/@beaucarnes/learn-the-mern-stack-by-building-a-n-exercisetracker- mern-tutorial-59c13c1237a1>

Documentation of Project Dependencies –

6. <https://nodejs.org/en/>
7. <https://www.mongodb.com/>
8. <http://expressjs.com/>
9. <http://mongoosejs.com/>
10. <https://aws.amazon.com/s3/>
11. <https://reactjs.org/> 12. <https://material-ui.com/>

Books –

13. Subramanian, Vasan. (2019). Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node. 10.1007/978-1-4842-4391-6.